

A Fast Disaster Recovery Mechanism for Volume Replication Systems

Yanlong Wang, Zhanhuai Li, and Wei Lin

School of Computer Science, Northwestern Polytechnical University,
No.127 West Youyi Road, Xi'an, Shaanxi, China 710072
{wangyl, linwei}@mail.nwpu.edu.cn, lizhh@nwpu.edu.cn

Abstract. Disaster recovery solutions have gained popularity in the past few years because of their ability to tolerate disasters and to achieve the reliability and availability. Data replication is one of the most key disaster recovery solutions. While there are a number of mechanisms to restore data after disasters, the efficiency of the recovery process is not ideal yet. Providing the efficiency guarantee in replication systems is important and complex because the services must not be interrupted and the availability and continuity of businesses must be kept after disasters. To recover the data efficiently, we (1) present a fast disaster recovery mechanism, (2) implement it in a volume replication system, and (3) report an evaluation for the recovery efficiency of the volume replication system. It's proved that our disaster recovery mechanism can recover the data at the primary system as fast as possible and achieve the ideal recovery efficiency. Fast disaster recovery mechanism can also be applicable to other kinds of replication systems to recover the data in the event of disasters.

1 Introduction

With the widespread use of computers, data is becoming more and more important in human life. But all kinds of accidents and disasters occur frequently. Data corruption and data loss by various disasters have become more dominant, accounting for over 60% [1] of data loss. Recent high-profile data loss has raised awareness of the need to plan for recovery or continuity. Many data disaster tolerance technologies have been employed to increase the availability of data and to reduce the data damage caused by disasters [2].

Replication [3] is a key technology for disaster tolerance and is quite different from the traditional periodical data backup. It replicates business data on a primary system to some remote backup systems in primary-backup architecture dynamically and on-line. It can not only retain the replicas in remote sites, but also make one of the backup systems take over the primary system in the event of a disaster. Therefore, it is widely deployed in disaster tolerance systems defend against both data loss and inaccessibility.

Disaster recovery mechanism (abbreviated to DRM) is one of the focuses in replication research fields. It helps us restore data after a disaster. Designing a right disaster recovery mechanism is an important and complex task. Although some

mechanisms have been presented, e.g. full or difference disaster recovery mechanism, we still need to improve the recovery efficiency and to afford the trade-off between data loss and the efficiency. It is ideal to implement a fast disaster recovery mechanism with the minimal data loss.

A replication system for disaster tolerance can be designed to operate at different levels, i.e. application-, file- or block-level. Of the three levels, block-level replication systems operate just above the physical storage or logical volume management layer, and they can take the advantage of supporting many different applications with the same general underlying approach. So we design a fast disaster recovery mechanism for volume replication systems at the block-level in this paper, although the mechanism could readily be implemented at file- and application-level.

The remainder of the paper is outlined as follows. In Section 2, we put our work into perspective by considering related work. In Section 3, we describe a general model of volume replication system and then describe two key disaster recovery mechanisms used in it. In Section 4, we introduce a fast disaster recovery mechanism, and discuss its principle and process in detail. In Section 5, we implement fast disaster recovery mechanism to recover data of the volume replication system on Linux and give a simple evaluation for it. Finally, we conclude the paper by highlighting the advantages of this work in Section 6.

2 Related Works

While more and more disaster tolerance systems are established, the study of disaster recovery mechanisms is motivated for practical importance and theoretical interest.

Many practitioners' guides (e.g., [4][5]) offer rules of thumb and high-level guidance for designing dependable storage systems, and recovering these systems after disasters. These books focus mainly on the logistical, organizational and human aspects of disaster recovery. They do not treat detailed disaster recovery issues for any certain technology.

Several recent studies explore how to improve the ability of a computer system or a storage system to recover from disasters and to meet user-defined goals (e.g., [6][7][8][9]). Practical guidelines are outlined that close the gap between business-oriented analyses and technical design of disaster recovery facilities in [6]. Some methods for automatically selecting data protection techniques for storage systems to minimize overall cost of the system are given in [7]. An adaptive scheduling algorithm is proposed for disaster recovery server processes in [8] and several methods for finding optimal or near-optimal solutions to deal with the data recovery scheduling problem are presented in [9]. All these studies focus on evaluating the risk of computer systems or the dependability of data storage systems and then making a best choice among various data protection techniques or making a best scheduling. Unfortunately, they do not give a detailed disaster recovery solution for a replication-based disaster tolerance system.

In the replication research literatures, we can find that replication protocols are the focus. Authors have proposed several protocols, including IBM's Peer-to-Peer Remote Copy (PPRC) [10], EMC's Symmetrix Remote Data Facility (SRDF) [11], Hitachi's Remote Copy [12] and VERITAS's Volume Replicator (VVR) [13]. Some

optimized replication protocols are presented, such as Seneca [14], SnapMirror [15] and RWAR [16]. Disaster recovery mechanisms are scarcely mentioned in these books and papers, and some of them simply introduce a planned Failback operation to restore each site to its pre-failure role.

Failback has already been researched in cluster computing [17] [18]. In order to build up a fault-tolerant cluster, the features of Failover and Failback services are demanded. Failback process is exercised when a system in the cluster is switched back from the normally secondary to the normally primary host. But Failback is mainly used to switch the role rather than to recover the data. So it is different between a cluster and a replication system.

To restore the data as soon as possible, we present a fast disaster recovery mechanism for volume replication systems based on referring to the technology of Failback. Our fast disaster recovery mechanism can not only switch the role of a replication system, but also restore its data fast.

3 Current Disaster Recovery Mechanisms for a Volume Replication System

3.1 The Volume Replication System

Volume management systems provide a higher-level view of the disk storage on a computer system than the traditional view of disks and partitions. This gives us much more flexibility in allocating storage to applications and users. A typical volume replication system is a remote and online replication system based on volume management systems. It is mainly composed of *Client*, *Primary* and *Backup* (or *Secondary*), and its architecture is shown in Fig. 1.

When *Primary* receives a request committed by *Client*, it writes a copy of the data of the request into the log volume and the storage volumes. At the same time, it sends the data to *Backup* and then *Backup* records the data.

Both *Primary* and *Backup* are a pair of peers to implement the process of replication. *Replicator* is their key component. It builds up the replication link

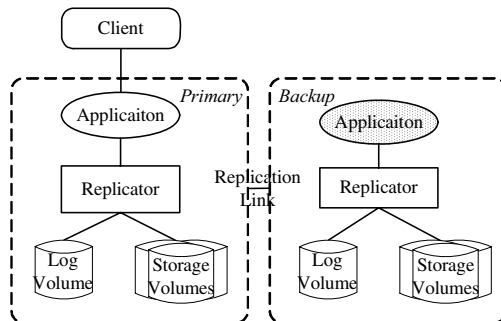


Fig. 1. Architecture of a typical volume system

between *Primary* and *Backup*, and runs the synchronous or asynchronous replication protocol. The log volume and the storage volumes are used to store the log and the data respectively. The volume replication system establishes one or more backup copies by replicating the data online and remotely. Once *Primary* suffers any disaster, *Backup* can take over the role of *Primary* and supply services to *Client*, which is referred to as Failover.

Obviously, the ability of the primary system (Primary) to tolerate disasters is increased by replicating. But if there is only a Failover mechanism, the primary system cannot restore its data according to the backup system (Backup) and be back to its primary role.

3.2 Current Disaster Recovery Mechanisms

Apart from the technology of Failback, there are some technologies to recover the data of the primary system, and they are unified as disaster recovery mechanisms. Full and difference disaster recovery mechanisms are the most key ones to recover the data of the primary system:

- Full disaster recovery mechanism: is a process to synchronize the former primary system with the current and new primary system (i.e. the former backup system) fully. That is, all the blocks at the former backup system are read and then sent to the former primary system. After receiving these blocks, the former primary system uses them to overwrite the local blocks.
- Difference disaster recovery mechanism: is a process to synchronize the former primary system with the current and new primary system by computing the differences between each pair of blocks at both the systems. That is, we compare a block at the former primary system with a corresponding one at the new primary system. If the pair of blocks are identical, the block at the new primary system does not need to be transferred to the former primary system; otherwise, the block must be sent to the former primary and be used to overwrite the local block. For all the blocks of the storage volumes at both the primary and backup systems have to be read, the differences are mostly gained by computing and comparing MD5 checksums for each pair of blocks to reduce the workload.

In order to describe the two mechanisms clearly, we define two rules as follow:

Definition 1: When restoring its data, a system must refer to the current and new primary system. It is defined as the newest referring rule (abbreviated to NRR).

Definition 2: After completing the process of restoring, a system must have the same image as the new primary system. It is defined as the data consistency rule (abbreviated to DCR).

At the same time, we define a set of symbols to describe the data sets at the different positions as follow:

Definition 3: D_{p-l} deontes the set of the log data at the primary system, D_{p-s} deontes the set of the storage data at the primary system, D_{b-l} deontes the set of the log data at the backup system, and D_{b-s} deontes the set of the storage data at the

backup system. D_{p-s} and D_{b-s} also denote the image of the primary system and the one of the backup system respectively.

If the storage volume is considered as a set composed of N data blocks, we can describe the process of full and difference disaster recovery mechanisms in Table 1.

Table 1. The processes of current disaster recovery mechanisms

Full disaster recovery mechanism	Difference disaster recovery mechanism
<pre> BEGIN i:=1; REPEAT B:=BackupReadBlock(i); BackupSendBlock(B); PrimaryRecieveBlock(B); PrimaryWriteBlock(B,i); i++; UNTIL i==N+1; END </pre>	<pre> BEGIN i:=1; REPEAT B_p:=PrimaryReadBlock(i); M_p:=ComputeMD5Checksum(B_p); PrimarySendMD5Checksum(M_p); BackupReceiveMD5Checksum(M_p); B_b:=BackupReadBlock(i); M_b:=ComputeMD5Checksum(B_b); Tag:=Difference(B_b,B_p); IF Tag=0 THEN BEGIN BackupSendBlock(B_b); PrimaryRecieveBlock(B_b); PrimaryWriteBlock(B_b,i); END END i++; UNTIL i==N+1; END </pre>

In Table 1, *Primary* denotes the former primary system and *Backup* denotes the former backup system (i.e. the new primary system). *BackupReadBlock()*, *BackupSendBlock()*, *PrimaryRecieveBlock()*, *PrimaryWriteBlock()*, *PrimaryReadBlock()*, *PrimarySendMD5Checksum()*, *BackupReceiveMD5Checksum()*, *ComputeMD5Checksum()* and *Difference()* are API functions to implement the process of disaster recovery mechanisms. It is easy for us to understand the former seven functions which involve four basic operations, i.e. *Read*, *Write*, *Send* and *Receive*. *ComputeMD5Checksum()* and *Difference()* are two important functions for difference disaster recovery mechanism. *ComputeMD5Checksum()* is used to compute the MD5 checksum for a block. *Difference()* is used to compare two MD5 checksums. If the two MD5 checksums are identical, it returns 1; otherwise, it returns 0.

As shown in Table 1, full disaster recovery mechanism needs to transfer all the blocks from the backup system to the primary system, so it is very time-consuming. Moreover, although difference disaster recovery mechanism reduces the number of transferred blocks, it needs to compare the MD5 checksums for each pair of blocks and then increases the number of the operations of *Read*, *Send* and *Receive*. In addition, when running full and difference disaster recovery mechanisms, users have to stop replicating the data of all applications. Therefore, it is possible for us to optimize both the disaster recovery mechanisms.

4 Fast Disaster Recovery Mechanism (FDRM)

4.1 The Principle of FDRM

To optimize the disaster recovery mechanisms, we present a fast disaster recovery mechanism. It tracks the incremental changes as they happen. Only the changed blocks at the new primary system are read and sent back to the former primary system. Hence the number of transferred blocks is smaller than full disaster recovery mechanism, and the number of read operations required is smaller than difference disaster recovery mechanism. In addition, it does not need to stop the replication operation of the replication system.

Fast disaster recovery mechanism synchronizes the former primary system with the new primary system by finding out the changed blocks at the new primary system after the former primary system broke down. That means that we need record the changes at the new primary system after it took over the former primary system. But some changes may be omitted. For example, in a volume replication system, some blocks have been wrote on the data volumes at the primary system before disasters occur to the primary system, but they may fail to be replicated to the backup system, be written on the data storage at the primary system, or even be acknowledged to the primary system after being written at he primary system. Thus the corresponding blocks at the backup system are stored as an older version than those at the primary system. According to NRR in Definition 1, when recovering the primary system, these blocks at the new primary system (i.e. the former backup system) become a part of all the changes.

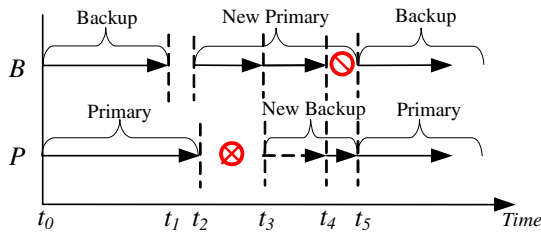


Fig. 2. Principle of fast disaster recovery mechanism

A simple description of fast disaster recovery mechanism is shown in Fig. 2. It assumes that P and B are a pair of systems for disaster tolerance. At first, P is the primary system and B is the backup system. At the time t_2 , P suffers some disasters and B takes over P . At that time, the image of P is $D_{p-s}(t_2)$ and the one of B is $D_{b-s}(t_2) = D_{b-s}(t_1)$. At the time t_3 , P comes back and becomes a new backup system for B . The data set of incremental changes at B from t_2 to t_3 , $(D_{b-s}(t_3) - D_{b-s}(t_2)) \cup (D_{b-s}(t_2) - D_{b-s}(t_1))$, is transferred back to P . This process can be implemented without stopping applications and last out from t_3 to t_4 , where

$D_{b-s}(t_3) - D_{b-s}(t_2)$ is the data set of real incremental changes at B from t_2 to t_4 and $D_{b-s}(t_2) - D_{b-s}(t_1)$ is the original data set of changes at P from t_1 to t_2 , $D_{p-s}(t_2) - D_{p-s}(t_1)$. Then, the data set of incremental changes at B from t_3 to t_4 , $D_{b-s}(t_4) - D_{b-s}(t_3)$, is transferred back to P . At the time t_5 , according to DCR in Definition 2, the whole recovery process is finished and P has the same image as B . P also can rerun as the primary system and supply services again.

4.2 The Process of FDRM

According the above description, the key issue for fast disaster recovery mechanism is to compute and gain $D_{b-s}(t_3) - D_{b-s}(t_2)$, $D_{b-s}(t_2) - D_{b-s}(t_1)$ and $D_{b-s}(t_4) - D_{b-s}(t_3)$. In order to simplify the problem, we build a change bitmap table for each storage volume and use it together with the log volume to track writes in the volume replication system. When a block in the storage volume at the primary system is changed, its data are also stored temporarily and a flag bit in the change bitmap table is set. Based on the bitmap tables at P and B , we can gain $D_{b-s}(t_3) - D_{b-s}(t_2)$, $D_{b-s}(t_2) - D_{b-s}(t_1)$ and $D_{b-s}(t_4) - D_{b-s}(t_3)$ easily.

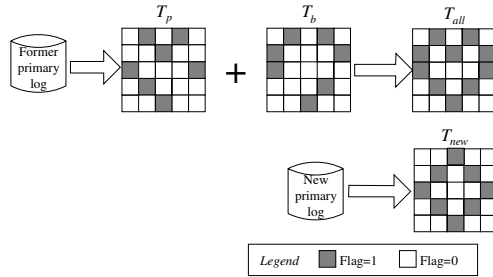
The process of fast disaster recovery mechanism is shown in detail in Table 2.

Table 2. The process of fast disaster recovery mechanism

Fast disaster recovery mechanism	
1	BEGIN
2	$T_p := \text{PrimaryBuildBitmapTable}(\text{Log}_p)$; //Build the bitmap table based on the log volume Log_p at the former primary system, and T_p denotes the bitmap table corresponding to the data set of $D_{p-s}(t_2) - D_{p-s}(t_1)$
3	$\text{PrimarySendBitmapTable}(T_p)$;
4	$\text{BackupReceiveBitmapTable}(T_p)$;
5	$T_{all} := \text{BackupComputeOR}(T_p, T_b)$; // T_b denotes the bitmap table for the data set of $D_{b-s}(t_3) - D_{b-s}(t_2)$ and is obtained by tracking the changes at the new primary system //Compute the value of OR operation for T_p and T_b , and T_{all} denotes the bitmap table for the data set of $(D_{b-s}(t_3) - D_{b-s}(t_2)) \cup (D_{b-s}(t_2) - D_{b-s}(t_1))$
6	$i := 1$; //Scan the bitmap table T_{all} , transfer the relevant data set to the former primary system and recover it
7	REPEAT
8	$\text{Flag} := \text{CheckBitmapTable}(T_{all})$;
9	IF $\text{Flag} == 1$ THEN
10	BEGIN
11	$B_b := \text{BackupReadBlock}(i)$;
12	$\text{BackupSendBlock}(B_b)$;
13	$\text{PrimaryRecieveBlock}(B_b)$;
14	$\text{PrimaryWriteBlock}(B_b, i)$;
15	END
16	$i++$;
17	UNTIL $i == N + 1$;

Table 2. (continued)

18	$T_{new} := \text{BackupBuildBitmapTable}(\text{Log}_b)$; //Build the bitmap table based on the log volume Log_b at the new primary system, and T_{new} denotes the bitmap table for the data set of $D_{b-s}(t_4) - D_{b-s}(t_3)$
19	$j := 1$; relevant // Scan the bitmap table T_{new} , transfer the data set to the former primary system and recover it
20	REPEAT
21	$\text{Flag} := \text{CheckBitmapTable}(T_{new})$;
22	IF $\text{Flag} == 1$ THEN
23	BEGIN
24	$B_b := \text{BackupReadBlock}(j)$;
25	$\text{BackupSendBlock}(B_b)$;
26	$\text{PrimaryRecieveBlock}(B_b)$;
27	$\text{PrimaryWriteBlock}(B_b, j)$;
28	END
29	$j++$;
30	UNTIL $j == N+1$;
31	END

**Fig. 3.** An example of building change bitmap tables

In the process of fast disaster recovery mechanism, the change bitmap table plays an important role. In Table 2, Step 2, Step 5 and Step 18 are the key steps to build T_p , T_{all} and T_{new} (T_b is build by tracking the changes from t_2 to t_3 at the primary system). An example is illustrated in Fig. 3. After T_p , T_b , T_{all} and T_{new} are computed, the data sets transferred to the former primary system are easy to obtain and then the former primary system can be recovered as fast as possible.

5 Implementation and Evaluation

Logical volume replicator (LVR) is a remote and online volume replication system prototype developed by us based on logical volume manager (LVM) [19] on Linux. We can create one replicated volume group at the primary system, which includes a log volume and several storage volumes. We also can create the other replicated volume group at the backup system. Both the replicated volume groups are a pair of peers to build up the replication link and implement remote replication and disaster recovery.

In order to evaluate the fast disaster recovery mechanism in detail, we compare it with full and difference disaster recovery mechanisms by analyzing theoretically and doing experiments.

5.1 Analysis

Fast disaster recovery mechanism can be evaluated with many parameters. Of all the parameters, the number of data blocks recovered, time consumption and space consumption are the three main ones. The number of data blocks recovered means

Table 3. Comparison of FDRM and conventional mechanisms

Mechanism	The number of data blocks recovered	Time consumption	Space consumption
Full disaster recovery mechanism	N	$N(t_r + t_n + t_w)$	$N \times s_d$
Difference disaster recovery mechanism	M	$N(2t_r + 2t_{md5} + \frac{S_m}{S_d}t_n + t_{diff}) + M(t_n + t_w)$	$N \times (2s_d + 2s_m)$
Fast disaster recovery mechanism	$I + J - K$	$I \times t_{bimap} + \frac{S_b}{S_d} \times t_n + t_{or} + (I + J - K)(t_r + t_n + t_w)$	$I \times s_d + 2s_b + (I + J - K)s_d$

how many data blocks are transferred to the former primary system to recover the data. Time consumption means how long it takes to implement the whole process of disaster recovery mechanism. And space consumption means the size of the memory space used at the primary and backup systems. The comparison of fast disaster recovery mechanism and the other two mechanisms are shown in Table 3.

In Table 3, we adopt some symbols to denote the element values of the three main parameters. They are explained as follow:

N is the number of the storage volume at the primary system or at the backup system, and M is the number of the different blocks at both the systems. I is the number of the blocks on the log volume at the primary system. These blocks were written to the storage volume but not replicated to the backup system, i.e. $I = |D_{p-s}(t_2) - D_{p-s}(t_1)|$. J is the number of the changed blocks at the backup system from the backup system taking over the primary system to now, i.e. $J = |D_{b-s}(t_3) - D_{b-s}(t_2)|$. K is the number of the same blocks between I blocks and J blocks, i.e. $K = |(D_{b-s}(t_3) - D_{b-s}(t_2)) \cap (D_{b-s}(t_2) - D_{b-s}(t_1))|$.

t_r , t_w and t_n are the time of *Read*, *Write* and *Transfer* respectively to deal with a block. *Transfer* includes two steps of *Send* and *Receive*. t_{md5} is the time to compute the MD5 checksum of a block. t_{diff} is the time to compare the MD5 checksum of one block at the primary system with that of the other corresponding block at the backup system. t_{bimap} is the time to establish a bitmap table according to the log at the

primary system. t_{or} is the time to compute the OR value of the primary bitmap table and the backup bitmap table.

In addition, s_d is the size of a data block. s_m is the size of a MD5 checksum. s_b is the size of a bitmap table.

From Table 3, we can find that the number of the recovered blocks in full disaster recovery mechanism is the largest one, and the number of the recovered blocks in different disaster recovery mechanism is equal or less than the one of fast disaster recovery mechanism. That is, $M \leq I + J - K \leq N$. We also can find that the time consumption and the space consumption in difference disaster recovery mechanism may be the largest of the three mechanisms because of computing the MD5 checksum for each block and comparing each pair of MD5 checksums. In addition, we can find that although the number of the recovered blocks in fast disaster recovery mechanism may be larger than different disaster recovery mechanism, the other two parameters in fast disaster recovery mechanism may be the smallest ones in all the three mechanisms.

When using MTBF (Mean Time Between Failures) and MTTR (Mean Time To Recover) to measure the reliability and the maintainability of the replication system, the failure rate λ is a constant for each disaster recovery mechanism, i.e. $MTBF = \frac{1}{\lambda}$ is a constant. At the same time, the maintenance rate μ in fast disaster recovery mechanism is the largest. Therefore, $Availability = \frac{MTBF}{MTBF + MTTR} \times 100\% = \frac{\mu}{\lambda + \mu} \times 100\%$ in fast disaster recovery mechanism is the largest and then it is the most ideal for a replication system.

5.2 Experiments

Obviously, it is not enough to compare the three disaster recovery mechanisms and to evaluate fast disaster recovery mechanism by analyzing. In order to understand the advantages of fast disaster recovery mechanism, we do some experiments based on LVR.

LVR is run on a pair of the primary and backup systems. Either of the two systems is an Intel Celeron 1.7 GHz computer with 512 MB of RAM. At the either system, the log volume is 1 GB and the storage volume is 2 GB. The versions of Linux kernel and LVM are 2.4.20-8 and 2.00.07 respectively. The block size used by LVM is set to 4KB and the Gigabit-Ethernet 1000baseT is used as connection between the primary system and the backup system. The replication protocol is the standard asynchronous one and the *random write* of IOzone-3.263 benchmark [20] is used to simulate the applications at the primary system.

With the cost of the memory becoming lower and lower, the space consumption is playing a less important role. So in the following two experiments, we do not consider the space consumption for the three mechanisms again.

In the first experiment, a manual disaster is made at a fixed time (i.e. t_2) after replication was started at the primary system. Disaster recovery occurs at a fixed time (i.e. t_3) after the backup system took over the primary system. Here t_2 is assumed to a moment when replication has been running for 100s and t_3 is assumed to a moment

Table 4. The number of recovered blocks and the time to recover for the three mechanisms

Mechanism	The number of data blocks recovered	The time to recover (s)
Full disaster recovery mechanism	524288	673
Difference disaster recovery mechanism	140243	464
Fast disaster recovery mechanism	173975	257

when the backup system has served as a new primary system for 30s. The results of comparing are shown in Table 4.

In the second experiment, we evaluate fast disaster recovery mechanism by testing the impact of t_3 . In practice, t_1 and t_2 are the objective moments that can not be changed artificially. t_4 and t_5 are the moments that are affected by t_3 . Here t_2 is still assumed to a moment when replication has been running for 100s and t_3 is the moment to start recovering the primary system where $t_3 - t_2$ is set with a series of values. The results are shown as Table 5.

Table 5. Impact of the moment to implement fast disaster recovery mechanism

$t_3 - t_2$ (s)	The number of data blocks recovered	The time to recover (s)
20	113821	127
40	320420	410
60	433975	543
80	500288	664
100	514337	729

From Table 4, we can find that the time to recover in fast disaster recovery mechanism is the smallest. From Table 5, we can find that it is ideal for fast disaster recovery mechanism to run as soon as possible after disasters occur.

In a word, it is proved by the above experiments that fast disaster recovery mechanism is worth deploying in the current replication systems.

6 Conclusion

With data becoming more and more important, all kinds of information systems require 24x7 availability. Replication is a rising data disaster-tolerance technique, and disaster recovery mechanisms play an important role in replication systems. How to recover the data at the primary system as soon as possible after the primary system suffers disasters is important to design a efficient replication system.

Fast disaster recovery mechanism is presented to recover the data at the primary system quickly and to improve the recovery efficiency of volume replication systems. Its principle and process are described in detail, and then it is implemented in logical volume replicator on Linux. It's proved by analyzing theoretically and doing experiments that fast disaster recovery mechanism makes the recovery process of volume replication systems fast and also guarantees the availability of the system. Therefore, it is helpful for volume replication systems to tolerate various disasters.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (60573096).

References

1. Smith, D.M.: The cost of lost data. *Journal of Contemporary Business Practice* 6(3) (2003)
2. Chervenak, A., Vellanki, V., Kurmas, Z.: Protecting file systems: A survey of backup techniques. In: *Proc. of Joint NASA and IEEE Mass Storage Conference, IEEE Computer Society Press, Los Alamitos* (1998)
3. Yang, Z., Gong, Y., Sang, W., et al.: A Primary-Backup Lazy Replication System for Disaster Tolerance (in Chinese). *Journal Of Computer Research And Development*, 1104–1109 (2003)
4. Cougias, D., Heiberger, E., Koop, K.: *The backupbook: disaster recovery from desktop to data center*. Schaser-Vartan Books, Lecanto, FL (2003)
5. Marcus, E., Stern, H.: *Blueprints for high availability*. Wiley Publishing, Indianapolis, IN (2003)
6. Cegiela, R.: Selecting Technology for Disaster Recovery. In: *DepCos-RELCOMEX 2006. Proc. of the International Conference on Dependability of Computer Systems* (2006)
7. Keeton, K., Santos, C., Beyer, D., et al.: Designing for Disasters. In: *FAST 2004. Proc. of the 3rd USENIX Conf on File and Storage Technologies*, pp. 59–72 (2004)
8. Nayak, T.K., Sharma, U.: Automated Management of Disaster Recovery Systems Using Adaptive Scheduling. In: *Proc. of the 10th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 542–545 (2007)
9. Keeton, K., Beyer, D., Brau, E., et al.: On the road to recovery: restoring data after disasters. In: *EuroSys. Proc. of the 1st ACM European Systems Conference*, pp. 235–248. ACM Press, New York (2006)
10. Azagury, A.C., Factor, M.E., Micka, W.F.: Advanced Functions for Storage Subsystems: Supporting Continuous Availability. *IBM SYSTEM Journal* 42 (2003)
11. Using EMC SnapView and MirrorView for Remote Backup. *Engineering White Paper, EMC Corporation* (2002)
12. *Software Solutions Guide for Enterprise Storage*. White Paper, Hitachi Data Systems Corporation (2000)
13. *VERITAS Volume Replicator 3.5: Administrator's Guide (Solaris)*. White Paper, Veritas Software Corp. (2002)
14. Ji, M., Veitch, A., Wilkes, J.: Seneca: remote mirroring done write. In: *USENIX 2003. Proc. of the 2003 USENIX Technical Conference*, pp. 253–268 (2003)
15. Patterson, H., Manley, S., Federwisch, M., Hitz, D., Kleiman, S., Owara, S.: SnapMirror: File-System-Based Asynchronous Mirroring for Disaster Recovery. In: *Proc. of the First USENIX conference on File and Storage Technologies* (2002)
16. Wang, Y., Li, Z., Lin, W.: RWAR: A Resilient Window-consistent Asynchronous Replication Protocol. In: *ARES 2007. Proc. of the 2nd International Conference on Availability, Reliability and Security*, pp. 499–505 (2007)
17. Hwang, K., Chow, E., Wang, C.-L., et al.: Fault-Tolerant Clusters of Workstations with Single System Image. In: *Cluster computing* (1998)
18. McKinty, S.: Combining Clusters for Business Continuity. In: *Cluster 2006. Proc. of the 2006 IEEE International Conference on Cluster Computing* (2006)
19. Redhat (2005), <http://sources.redhat.com/lvm2/>
20. IOzone (2006), <http://www.iozone.org/>